Lecture 4

Monday, September 30, 2024 9:22 AM

Housekeeping

Remind me to share screen, start recording!

- HW 1 Solutions coming soon
- HW 2 posted; due in < 2 weeks
- Project Part 1; posted due in < 3 weeks

Today:

- Loose ends:
  - RA: rename, assignment, generalized RA (?)
  - TRC examples
- Continue SQL
- Time permitting, some HW 1 solutions

Rename operation

Monday, February 13, 2023 9:26 AM

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example: $\rho_x(E)$

  returns the expression E under the name X
- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1,A_2,\ldots,A_n)}(E)$$

  returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.
- E.g.. Given relation CAKE(cname, price, slices)

$$\rho_{cheapcake(cakename, price, slices)}\ (\sigma_{price < 10}(CAKE))$$

Database System Concepts - 6th Edition
Modified by T. Suel for CS6083, NYU Tandon, Spring 2022 2.1 ©Silberschatz, Korth and Sudarshan

\ tmp ← ρcheapcake(. . . .)

RA formal def

Monday, February 13, 2023 9:50 AM

Formal Definition

A basic expression in the relational algebra consists of either one of the following:

- A relation in the database
- A constant relation

- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p(E_1)$, P is a predicate on attributes in E_1
  - $\Pi_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x(E_1)$, x is the new name for the result of E_1

RA, SQL, TRC Examples

Monday, February 13, 2023 9:48 AM

Example 1 RA, SQL

Thursday, February 9, 2023 11:45 AM

Foo

num	letter	color
1	A	red
2	B	blue
3	C	green

Find colors of foos with num ≥ 2

Relational Algebra: $\Pi_{color}\ \sigma_{num \geq 2}\ (foo)$

SQL:

```
SELECT color
FROM foo
WHERE num >= 2
```

Example 1 TRC

Thursday, February 9, 2023 11:45 AM

Foo

num	letter	color
1	A	red
2	B	blue
3	C	green

Find colors of foos with num ≥ 2 :

TRC : Find $t[color]$ s.t

there is a row $u \in foo$

for which $u[num] > 2$

and $t[color] = u[color]$

$$\{ t \mid \exists u \in foo \, (u[num] > 2 \wedge t[color] = u[color]) \}$$

Result

color
blue
green

Example 2 RA/SQL

Thursday, February 9, 2023 11:45 AM

Foo

num	letter	color
1	A	red
2	B	blue
3	C	green

Bar

color	shape
blue	square
blue	triangle
green	circle

Find shapes of bars for which there is a foo of same color with num = 2

Relational Algebra:

$$\Pi_{shape}\ \sigma_{num=2}\ (foo \bowtie bar)$$

SQL

```
SELECT shape
FROM foo NATURAL JOIN bar
WHERE num = 2
```

Example 2 TRC

Thursday, February 9, 2023 11:45 AM

Foo

num	letter	color
1	A	red
2	B	blue
3	C	green

Bar

color	shape
blue	square
blue	triangle
green	circle

Find shapes of bars for which there is a foo of same color with num = 2

Domain Relation Calculus

$\{ t \mid$

Square ? Yes $\langle$blue, square$\rangle \in$ bar 1

$\langle 2, B, \text{blue} \rangle \in$ foo

<2, 13, blue> ∈ foo

triangle ? Yes <blue, triangle> ∈ bar

∧ <2, 13, blue> ∈ foo

circle ? No

SELECT attribute-list

FROM expression-representing-relation

WHERE predicate

Joins

- For all instructors who have taught courses, find their names and the course ID of the courses they taught.

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

Join condition

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
select section.course_id, semester, year, title
from section, course
where section.course_id = course.course_id and
      dept_name = 'Comp. Sci.'
```

- Note: you almost always want a join, not a Cartesian product. Also, joins are usually done along foreign keys.

Monday, February 6, 2023 10:59 AM

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **select** *
  from *instructor* **natural join** *teaches*;

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

- Note: in general, join can be specified explicitly or in where clause.

Join conditions

Monday, February 13, 2023 9:43 AM

NATURAL JOIN – match on all common attributes

JOIN USING (A1, A2, …) ⋈ match on these common attributes

JOIN ON (predicate)

Names of students & IDs of their advisors:

student(ID, name, dept_name, tot_cred)

advisor(s_id, i_id)

```
SELECT name, i_id
FROM student, advisor
WHERE ID = s_id
```

— Can't use NATURAL join bc the attributes have diff names

```
SELECT name, i_id
FROM student JOIN advisor
ON (ID = s_id)
```

student.ID = advisor.s_id

Tuesday, September 27, 2022 7:43 PM

Find names of students & their advisors (names of)

```
SELECT student.name, instructor.name
FROM (student JOIN advisor ON
      (student.ID = s_id))
     JOIN instructor ON
      (instructor.ID = i_id)
```

student (ID, name, dept_name, tot_cred)
advisor (s_id, i_id)
instructor (ID, name, dept_name, salary)

(should not match)

Monday, February 6, 2023 10:59 AM

Natural Join (Cont.)

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly
- List the names of instructors along with the the titles of courses that they teach
- Incorrect version (equates course.dept_name with instructor.dept_name)
  - **select** *name*, *title*
    from *instructor* **natural join** *teaches* **natural join** *course*;
- Correct version
  - **select** *name*, *title*
    from *instructor* **natural join** *teaches*, *course*
    where *teaches.course_id= course.course_id*;
- Another correct version
  - **select** *name*, *title*
    from (*instructor* **natural join** *teaches*) **join** *course* **using**(*course_id*);

NOT matching dept-name

Better syntax IMO

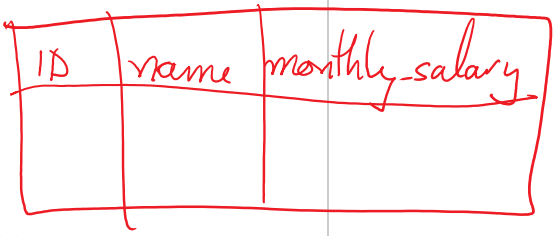
Monday, February 6, 2023 10:59 AM

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- E.g.,
  - **select** *ID, name, salary/12* **as** *monthly_salary*
    from *instructor*



- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - **select distinct** *T. name*
    from *instructor* **as** *T, instructor* **as** *S*
    where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*
- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

- *Note: some systems require* ***as*** *to be omitted in* ***from*** *clause.*

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all instructors whose name includes the substring "dar".

```
select name
from instructor
where name like '%dar%'
```

- Match the string "100 %"

```
like '100 \%' escape '\'
```

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Monday, February 6, 2023 10:59 AM

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors
  select distinct *name*
  from *instructor*
  order by *name*
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

lexicographic order

$(A, B) < (C, D)$ iff

1) $A < C$ or
2) $A = C$ and $B < D$

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000
- Tuple comparison
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, ’ Biology’);

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:
  1. $\sigma_\theta(r_1)$: If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$.
  2. $\Pi_A(r)$: For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.
  3. $r_1 \times r_2$: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are $c_1 \times c_2$ copies of the tuple $t_1 . t_2$ in $r_1 \times r_2$

Duplicates (Cont.)

- Example: Suppose multiset relations r_1 (A, B) and r_2 (C) are as follows:

  r_1 = {(1, a) (2,a)} r_2 = {(2), (3), (3)}

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1)$ x r_2 would be

  {(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)}

- SQL duplicate semantics:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1,A_2,\square,A_n}(\sigma_P(r_1 \times r_2 \times \square \times r_m))$$

Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  union
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 and in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 but not in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

Not supported by MySQL

oracle : except → minus

Set Operations

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.

- Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:
  - *m* + *n* times in *r* **union all** *s*
  - min(*m*,*n*) times in *r* **intersect all** *s*
  - max(0, *m* – *n*) times in *r* **except all** *s*

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null
```

Monday, February 6, 2023 10:59 AM

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - Example: *5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*,
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

P	q	∧	∨
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F
T	U		
F	U		
U	T		
U	F		

Aggregate Functions

- These functions operate on the multiset of (numerical) values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

(Note: not expressed in Relational Algebra / or Relational Calculus
are in extended Relational Algebra)

Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department
  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= ' Comp. Sci.' ;
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *semester* = ' Spring' **and** *year* = 2010
- Find the number of tuples in the *course* relation where dept_name = 'Comp Sci'
  - **select count** (*)
    from *course*;
    where dept_name = 'Comp. Sci.'

Monday, February 6, 2023 10:59 AM

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*) AS avg_salary
    from *instructor*
    group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list
  - /* erroneous query */
    select *dept_name*, *ID*, **avg** (*salary*)
    from *instructor*
    group by *dept_name*;

- Note: a very common mistake. But note that having ID in here also does not make sense if you think about it.
- But how about which instructor has highest salary in each department?

  select *dept_name*, *ID*, **max** (*salary*)
  from *instructor*
  group by *dept_name*;

  - Still not allowed!
  - And there could be several instructors making the maximum

- VIOLATES RULE
- MAY GIVE GARBAGE

Aggregation (Cont.)

- And ***never never*** do this!
  - /* erroneous query */
    select *ID*
    from *instructor*
    having max(salary)
- Does this output instructor making the maximum?
- No, completely nonsensical
- max(salary) is a number, say 20000.
- What does “having 20000” mean?
- having-expression has to evaluate to true of false

Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Aggregate Functions – Having Clause

- Full version of aggregation queries:
- select ... from ... where ... group by ... having ...
- E.g.,

```
select dept_name, avg (salary)
from instructor join department
where budget > 100000
group by dept_name
having avg (salary) > 42000;
```

Note: this is equivalent to RA expression

$$\Pi_{dept\_name,\ avs}\ (\ \sigma_{\ avs > 42000}\ (_{dept\_name}\mathcal{G}_{\mathbf{avg}(salary)\ as\ avs}\ (\sigma_{budget > 100000}\ (instructor \bowtie department\ )\ )\ )\ )$$

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null
- Note: sum/count is not always same as avg, due to null values

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
      course_id in (select course_id
                    from section
                    where semester = 'Spring' and year= 2010);
```

in ∈

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
      course_id not in (select course_id
                        from section
                        where semester = 'Spring' and year= 2010);
```

not in ∉

Example Query

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                     from teaches
                     where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

Usually should not do this:

- Output the names of all instructors who are in departments with budget > 100000

```
select name
from instructor
where dept_name in (select dept_name
                    from department
                    where budget > 100000);
```

- This is correct syntax, but a very complicated way to do a simple join
- Instead just do a join:

NATURAL

```
select name
from instructor join department
where budget > 100000;
```


n

Set Theory operations

Tuesday, March 10, 2020 9:38 PM

Set theory operation	**SQL Predicate**
Union ∪	UNION [can often be done with pred1 OR pred2 in the WHERE clause, instead]
Intersection ∩	INTERSECT [Not supported in MySQL]
Set Difference −	EXCEPT [MINUS in Oracle; not supported in MySQL]
Belongs to ∈	IN (subquery describing X) ∈
Does not belong to ∉	NOT IN (subquery) ∉
Is empty = ∅	NOT EXISTS (subquery)
Is not empty ≠ ∅	EXISTS (subquery)
Comparison to each element	ALL/SOME operator (subquery)

Comparison of sets to each other:

X subset Y — important operations, but not supported directly ☹

Must rewrite into complicated form

$X - Y = \emptyset \Leftrightarrow X \subseteq Y$

$X == Y$ — similar issues, even messier

Set membership

Tuesday, March 10, 2020 9:41 PM

predicate (usually in WHERE clause):

<attribute-list> IN (SELECT <attribute-list> FROM… WHERE …)
IMPORTANT: attribute lists must be same (compatible)

Saw

userID	title	Release_yr	stars
12345	Little Women	2019	5
12345	Little Women	1994	4
54321	Little Women	2019	4
54321	Finding Dory	2016	3
67890	Little Women	2019	5
67890	Finding Dory	2016	3

Find people who saw Little Women(2019) and Finding Dory (2016)

```
SELECT userID FROM Saw
WHERE title = 'LW' AND releaseYr = 2019
AND userID IN (
SELECT id FROM Saw
WHERE title = 'FD' AND relYR = 2016 )
```

predicate

Find people who saw Little Women (2019) and did not see Finding Dory (2016)

IN → NOT IN

Set difference

Aside: intersections with joins:

Tuesday, March 10, 2020 9:50 PM

Saw 1

userID	title	Release_yr	stars
12345	Little Women	2019	5
12345	Little Women	1994	4Set operations revisited
54321	Little Women	2019	4
54321	Finding Dory	2016	3
67890	Little Women	2019	5
67890	Finding Dory	2016	3

Saw 2

userID	title	Release_yr	stars
12345	Little Women	2019	5
12345	Little Women	1994	4Set operations revisited
54321	Little Women	2019	4
54321	Finding Dory	2016	3
67890	Little Women	2019	5
67890	Finding Dory	2016	3

Find people who saw Little Women(2019) and Finding Dory (2016)

```
SELECT userID
FROM   Saw AS Saw1 JOIN Saw AS Saw2
       USING (userID)
WHERE  Saw1.title = 'LW' AND
       Saw1.Release_yr = 2019
AND    Saw2.title = 'FD'
AND    Saw2.Rel_yr = 2016
```

Scalar subqueries

Tuesday, March 10, 2020 9:53 PM

- Query that is guaranteed to return a single row (no matter what valid data is in the database)
- Can be used like a value of the returned type, e.g.
  - Comparison with <, >, =, etc
  - Arithmetic operations if it's a number
- Can be used in comparisons to result of aggregation
  - Find the movie(s) that got the highest rating (from someone)

userID	title	Release_yr	stars
12345	Little Women	2019	5
12345	Little Women	1994	4
54321	Little Women	2019	4
54321	Finding Dory	2016	3
67890	Little Women	2019	5
67890	Finding Dory	2016	3

DISTINCT

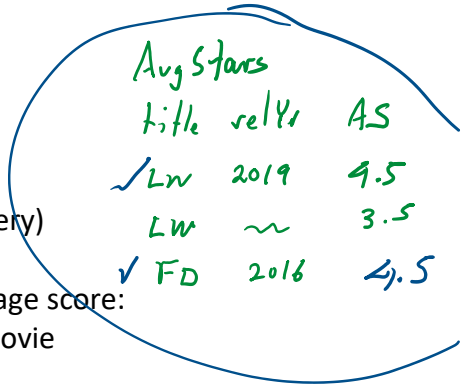
```
SELECT title, Release_yr, MAX(stars)
FROM Saw
```

Remember that attributes in SELECT must be aggregated or in GROUP BY

(1) Find max — Scalar subquery
(2) " movies that got that score

VIEWS

Tuesday, March 10, 2020 9:58 PM



CREATE VIEW view_name AS (query)

Find movie**s** with the highest average score:

1. Find average score of each movie
2. Find max of those
3. Find title/year of the movie whose average is that max

```
CREATE VIEW avgStars AS
  ( SELECT title, relYr, AVG(numStars) AS avs
    FROM Saw
    GROUP BY title, relYR )

SELECT title, relYR
FROM avgStars
WHERE avs = (SELECT max(avs)
             FROM avgStars)
```

① ② ③